A Cloud based

Continuous Delivery Software Developing System

on Vlab Platform

by

Yuli Deng

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2013 by the
Graduate Supervisory Committee:

Dijiang Huang, Chair
Hasan Davulcu
Yinong Chen

ARIZONA STATE UNIVERSITY

December 2013

ABSTRACT

Continuous Delivery, as one of the youngest and most popular member of agile model family, has become a popular concept and method in software development industry recently. Instead of the traditional software development method, which requirements and solutions must be fixed before starting software developing, it promotes adaptive planning, evolutionary development and delivery, and encourages rapid and flexible response to change. However, several problems prevent Continuous Delivery to be introduced into education world.

Taking into the consideration of the barriers, we propose a new Cloud based Continuous Delivery Software Developing System. This system is designed to fully utilize the whole life circle of software developing according to Continuous Delivery concepts in a virtualized environment in Vlab platform.

DEDICATION

To My Beloved Family.

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor and committee chair Dr. Dijiang Huang for his invaluable guidance, advice and support. I also would like to thanks Dr.Hasan Davulcu and Dr. Yinong Chen for their great feedback and advice to my thesis work as my thesis committee.

Having worked in the Secure Networking and Computing (SNAC) Lab for last two years, I also really want to thanks every member of the lab for their help and support to me and make this all possible.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Recently, in software development industry, Continuous Delivery has become a popular concept and method. Traditional software development method is based on the idea that a team should start developing software only when all the design of the software is done and well documented and should only making software ready for the release when all the functionality for the release has been developed. Agile development, on the other hand, introduces the idea that software development should be an iterative and incremental development process, requirements and solutions should evolve through collaboration between teams during the development process. It suggests adaptive planning, evolutionary development and delivery, and requires rapid and flexible response to change [1]. Continuous Delivery, is a subset of agile development which in which the team spends efforts to keeps the software they developed ready for release any times during development. It is different from traditional agile development by that the effort to create a release is done by an automated system. Techniques such as automated testing, continuous integration and continuous deployment allow software to be developed with a high quality and can be easily warped and deployed to production environments [2]. And as a result, it also provide developer with the ability to push out enhancements and bug fixes to customers rapidly, reliably and repeatedly at low risk and with very small overhead. This project attempts to adopt the Continuous Delivery concepts on to Vlab platform. Vlab platform is a cloud based resource and service sharing platform for computer education initially established by Aniruddha Kadne for his master thesis during the summer of 2010 in SNAC Lab in Arizona State University [3]. The Vlab system is still evolving and has

been redesigns and redeveloped for several times since then. It is now used by class instructors as a laboratory environment for students for multiple classes in Arizona State University [4].

By adopting the Continuous Delivery concepts on to Vlab platform, we aim to provide instructors and students an automated software development infrastructure with low learning curve and high availability based on Cloud environment. To achieve this result, this research also aims at developing a user-friendly, easy-to-use web user interface that enable users to do the software development using Continuous Delivery totally remotely.

1.1 Motivation of the Research

High level software engineering education has become very popular in the past years. A great number of universities provided Software Engineering degree programs; as of 2010, there were 244 Campus programs, 70 online programs, 230 Masters-level programs, 41 Doctorate-level programs, and 69 Certificate-level programs in the United States [5]. But it is still a well-known fact that United States is on short of software engineers. The main reason induced this problem is that there is a large gap between what students learnt in classroom and what is needed in current software industry.

Among other knowledge and skills, software development skill is one of the most important area for undergraduate computer engineering education since knowledge of software development is a pre-requisite to becoming a software engineer. And the most important part of learning software development skill is examination and experimentation with the skill they learnt in class like doing a software development project. But according to our survey and analysis, this is where the gap lies.

Mainstream software engineering education nowadays are still mainly focus on traditional software development process models like waterfall model. These models are all sequential models. It proceeds to the next phase only when the current phase is done. This is exactly the same as those sequential design process in traditional manufacturing industries, and as a result, it is always very costly to make changes after the design part is done, if not impossible. This is the reason why in current software industry, another set of software development models called agile model became very popular. Agile software development uses iterative development as a basis but advocates a lighter and more people-centric viewpoint than traditional approaches. Agile processes fundamentally incorporate iteration and the continuous feedback that it provides to successively refine and deliver a software system.

Continuous Delivery, as one of the youngest and most popular member of agile model family, have been proven to have many advantages:

- Immediate unit testing, low level bugs can be found immediately after an integration.
- Immediate feedback, system will notify developer once there is bug found in the lasted change of code.
- Early warning of broken/incompatible code and conflicting changes.
- Ability of roll back to a bug-free version when there is a bug emerges.
- Detect and fix integration problems continuously.
- Constant availability of a latest build for testing, demo, or release purposes.

But there are also barriers for this kind of agile development model to be introduce into classroom. Some of the problems comes from Continuous Delivery itself. Continuous Delivery has some certain disadvantages make it hard for students to apply it to a software development assignment or a semester long course project.

First problem is that the initial setup time required for a Continuous Delivery project is relatively longer than traditional software development project. The reason is simple. For traditional software development, one student just need to install some well-developed IDE and then he can start programing. But for a Continuous Delivery project, multiple tools and components need to be setup to automate the whole development process before programing. The detail information of the tools and components need for Continuous Delivery are provided in Chapter 3.

Second problem is that well-developed test script is required to achieve automated testing advantages of Continuous Delivery. This is normally hard for students since most of them are still learning how to develop software and know nothing about how to design tests to test their own software.

Third problem is that the initial setup costs of hardware in Continuous Delivery can be significant. Even for a small project, one student will still need a standalone server besides their own computer.

1.2 Proposed Solution

Taking into the consideration the above mentioned problems, we propose a new Cloud based Continuous Delivery Software Developing System. This system is designed to fully utilize the whole life circle of software developing according to Continuous Delivery concepts in a virtualized environment in Vlab platform for educational purpose. Every part of the life circle will be handle by one or multiple virtual machines and the cloud system will monitor the resource workload and available physical resources to provide resource dynamically and make all these software development services on a continual basis. Another important aspect of this research is to make this platform available to students anywhere and anytime by Cloud. Class instructors can set up software development template in the system for certain software development assignment or project. Then students can simply upload their software source code and the system will compile, test, and package the source code in to software package or other deliverable material according to the preset template and can even do functional assessment and evaluation for instructors.

CHAPTER 2

VLAB PLATFORM BACK GROUND

In this section the information and technical detail about Vlab platform will be provided. As described in previous chapter, Vlab platform is the backbone of the proposed Continuous Delivery Software Developing System. Vlab platform is a cloud-based virtual laboratory environment established on the MobiCloud Platform. It is designed to provide a virtual computer network environment for students with full control of network nodes from the physical layer to the application layer, this system can greatly reduce network establishment overhead and provide desired levels of fidelity for course project experiments. The Vlab adopts a set of virtualization techniques to implement a customizable, reconfigurable, and isolated real computer networking system. The Vlab's crowdsourcing platform enables interactive and collaborative learning. It will help the course material collection, classification, and filtering, and it can be easily customized for personal as well as for groups of users [4]. In this section it will introduce current Vlab architecture and each component of current Vlab platform.

2.1 Vlab Platform Architecture

The current Vlab platform is based on OpenStack project. The OpenStack project as a whole is designed to "delivering a massively scalable cloud operating system." To achieve this, each of the constituent services are designed to work together to provide a complete Infrastructure as a Service. This integration is facilitated through public application programming interfaces (APIs) that each service offers. These APIs allow each of the

services to use another service. These are (mostly) the same APIs that are available to end users of the cloud [6].

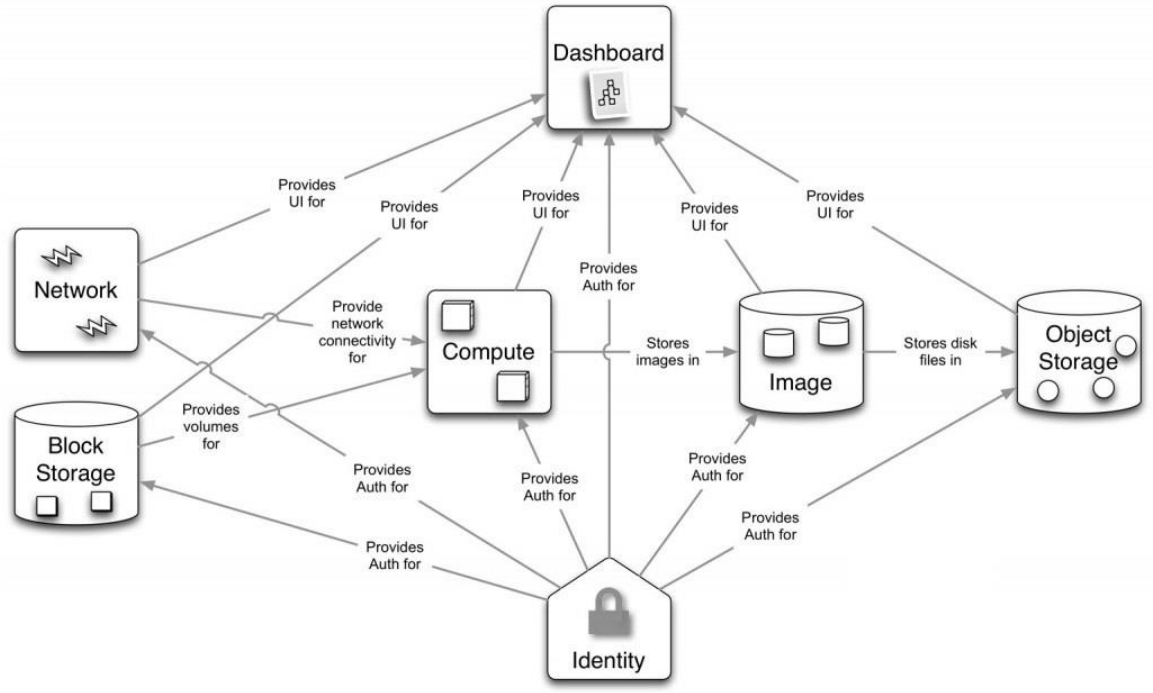


Figure 2-1: Vlab Platform Architecture

Figure 2-1 shows the architecture and relationships between different components.

- Dashboard provide a web front UI for all other services in the Vlab platform.
- Compute is where all VMs runs, it retrieves and stores images and other data in the Image part.
- Network provides virtual networking and network management for Compute.
- Image stores the actual virtual disk files in the Object Storage.
- Identity does all the services authenticate.

More detail information about Vlab platform can be found in Appendix A which briefly describes the features and capabilities of Vlab platform. Readers interested in understanding only the workflow and analysis of this research, can very well skip that part.

CHAPTER 3

CONTINUOUS DELIVERY

In this section it will introduce basic concepts of Continuous Delivery, Continuous Delivery Pipeline structure, Continuous Delivery lifecycle and the benefits of using Continuous Delivery.

3.1 Concepts of Continuous Delivery

Continuous Delivery is all about automation of the software development process. It is a series of practices designed to ensure that code can be rapidly and safely deployed to production by delivering every change to a production-like environment and ensuring business applications and services function as expected through rigorous automated testing [7]. It combines several core concepts including continuous integration, continuous testing, continuous deployment and continuous feedback.

3.1.1 Continuous Integration

Continuous integration means a software development practice where members of a software development team integrate their work on a frequent basis. Each release of software shall be verified by an automated build and test to detect problem as frequently as possible. To achieve this, certain automated environments for builds need to be introduced in to the build process.

3.1.2 Continuous Testing

Continuous testing is to include automated tests into the build process. Continuous testing can automate some repetitive but necessary tasks in a formalized testing process already in place, or add additional testing that would be difficult to perform manually. For self-testing code, a suite of automated tests that can check a large part of the code base for bugs is needed. The tests need to be able to run from a simple command and to be self-checking. The result of running the test should indicate if any tests failed.

3.1.3 Continuous Deployment

Continuous deployment is basically keep publishing software and put software into production every time when it pass the continuous testing part. It is a set of practices and steps which enable software developers to release software any time, any place automatically.

3.1.4 Continuous Feedback

Continuous feedback is the key output of a Continuous Delivery system. Since Continuous Delivery is a totally automated process, information about the result of every integration is very important to developers. Developer want to know as soon as possible if there was a problem with the latest build. Without Continuous feedback, none of the other part of Continuous Delivery is useful.

3.2 Continuous Delivery Pipeline

With all the concepts of Continuous Delivery in mind, the next step of implement a Continuous Delivery system is to design a Continuous Delivery pipeline.

The purpose of Continuous Delivery pipeline is to automate software compile, test, deploy and publish process. Every source code commit operation triggers a number of checks, which give the development team an immediate feedback on the software quality. This part of the CDP functionality covers processes which are typically associated with a continuous integration process. Furthermore, the software is immediately and automatically deployed and tested in development and pre-production environments. Finally, the release of a selected software version is triggered by one mouse click which is safer and quicker compared to custom software releases, where the operations team needs to follow a word document to get the next release into production.

The structure of a CDP is derived from the project value stream from check-in to release. Definition of a suitable value stream setup is therefore the most vital task for setting up a CDP structure. Usually one software developer needs a couple of iterations to find an appropriate CDP structure which suits the given project.

As a result, the structure of a CDP will be quite different from software project to software project. But most of them follows the basic lifecycle of Continuous Delivery in the next section.

3.3 Continuous Delivery Lifecycle

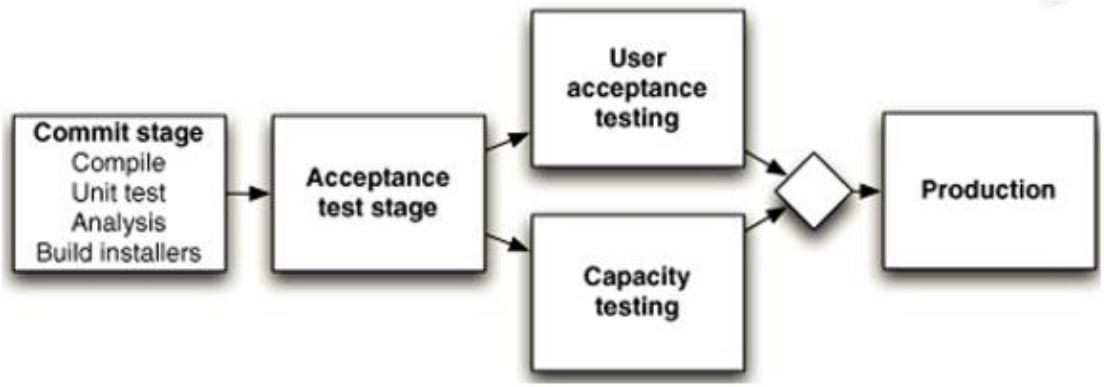


Figure 3-1: Continuous Delivery Lifecycle

A basic Continuous Delivery Lifecycle involves the stages shows in the Figure 3-1. All stages will be introduced one by one in the following subsections.

3.3.1 Commit Stage

This stage pull the latest committed code from the version control system and evaluate it. CI machine will do the following jobs and get results:

Code compilation: CI machine will compile the code from version control system and if this job fails, CI machine will send out emails immediately to notify the development team.

Unit tests: Unit tests written by developer will be run by the CI machine. These tests focus on a low API level and are designed to check basic functionality of the software.

Versioning: To manage versioned software binaries, the CI machine will tag the source code and push them back into the version control system after compilation and unit testing.

Static source code quality check: This is an optional job done by some quality checking tool.

3.3.2 Acceptance Test Stage

Automated acceptance test stages aims to test the system works on the functional and logic level, that behaviorally it follows the design of the development team. This makes the automated acceptance tests stage to be a user-oriented testing stage by testing user needs and requirements on the functionality level. To achieve the goal, the software for acceptance testing needs to be deployed into a close-to-production environment. By provision cloud based virtual environment to a production environment, deploying the software and having the acceptance tests done, the software will be proved to be fully functional.

3.3.3 User Acceptance Test (UAT) Stage

User acceptance test stage is similar to automated acceptance test stage, the CI machine prepares the same environment closed to the production environment. The difference is that the testing is done by development team instead of automated testing. In the testing environment, the team can test those function which cannot or is very hard to be checked by automated tests like very complex work-flows in the software. So that all the tests in this stage are done manually.

3.3.4 Capacity Test Stage

If both user acceptance test stage and automated acceptance test stage have been successfully executed, continuous delivery pipeline will package a ready-to-publish software. In some cases, the software can be delivered immediately, but for some other

cases, however, other non-functional parameters must be tested too. For example, for a web application, responsiveness and stability under a heavy workload must be tested

By using stability and workload testing tools like JMeter [15], this stage will check whether the latest code and configuration affect application performance thresholds.

3.3.5 Delivery Stage

As the last stage of the lifecycle, delivery stage delivers the system to users, either as packaged software or by deploying it into a production environment. Since the versioning has been already done stored in the version control repository, any stored software version can be delivered by a single trigger by development team. What's more, delivery stage in the Continuous Delivery pipeline can also be automated, which means the delivery can be triggered by the CI machine automatically.

3.4 Benefits of Using Continuous Delivery

After understand the basic concepts and working flow of a Continuous Delivery system, it will evaluate the benefits of using a Continuous Delivery system in this section.

The most important and wide ranging benefits of Continuous Delivery is reduced risk [8]. In traditional software development project, software developers only do a deferred integration when their parts of work are totally done by themselves separately. The problem with this practice is that it's very hard to predict how long and how much of effort it will take to get the integration done.

Continuous delivery completely avoid this problem. Since integration is done continuously on a daily basis, software developer always know when there is a problem, what works, what doesn't.

Continuous delivery doesn't get rid of bugs, but it does make them dramatically easier to find and remove. In this respect it's rather like self-testing code. If one software developer introduce a bug and detect it quickly it's far easier to get rid of. Since there is only a small bit of the code has been changed from the last working integration, it's relatively easy to locate the bug. Also, since the change is just done by the developer, he's totally aware of what changes he have made before he forget these. By using diff debugging - comparing the current version of the system to an earlier one that didn't have the bug, it is even simple for other developer to find where the problem is.

Bugs are also cumulative. The more bugs there are, the harder it is to remove each one. This is partly because there are bug interactions, where failures show as the result of multiple faults - making each fault harder to find. It's also psychological - people have less energy to find and get rid of bugs when there are many of them.

As a result, projects with Continuous Delivery tend to have dramatically less bugs, both in production and in process.

Another benefit of Continuous Delivery is that it removes one of the biggest barriers to frequent deployment. Frequent deployment is valuable because it allows end users to get new features more rapidly, to give more rapid feedback on those features, and generally become more collaborative in the development cycle.

CHAPTER 4

SYSTEM DESIGN

In this section, it will first discuss the overall architecture of the proposed Continuous Delivery software developing system on Vlab Platform, then explains each components in the systems and end up with description of several usage scenarios to explain how the system works.

4.1 Overall Architecture

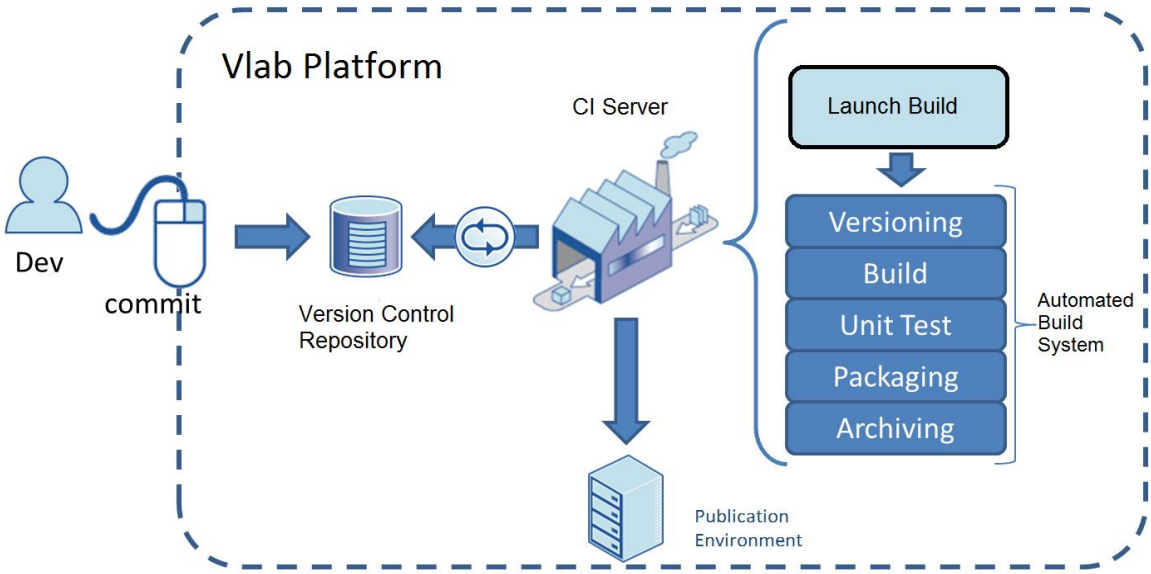


Figure 4-1: System Overall Architecture

Figure 4-1 shows a simplified architecture of the proposed Continuous Delivery software developing system. The Architecture are consisted of 3 main components, Version Control Repository, CI Server and Automated Build and Test System. Some other core components of the system include Build Script used by Automated Build and Test System and Feedback Mechanism is contained in the CI Server.

4.1.1 Version Control Repository

A Version Control Repository keeps track of all work and all changes in a set of files like source code and other software, and allows members of a software development team to work together on the same set of code [9]. The purpose of using a version control repository in Continuous Delivery is to keep the track of changes to codes and provides a single point of access so that all source codes are available from one location by all the team members. It also allows roll back function so that you can get old version of codes or other files any time.

There are normally two kinds of version control repository for computer programing, centralized version control repository and distributed version control repository [10].

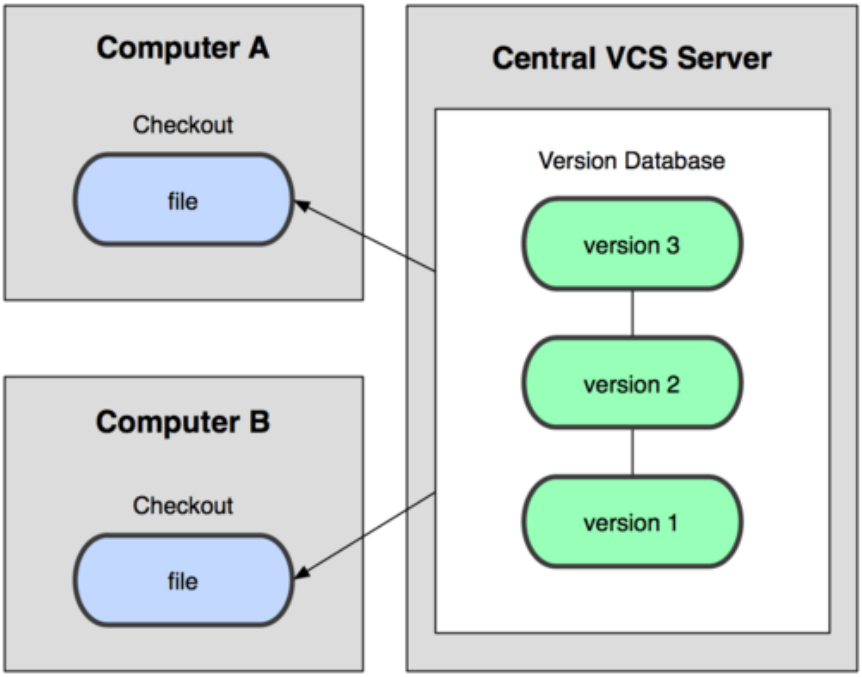


Figure 4-2: Centralized Version Control Diagram

Centralized version control repository, as its name stated, store all the files in the repository at a centralized place. The system works on a client-server approach as showed in Figure

4-2, every developer run a client locally to communicate with server side where the only copy of files is located. This type of setup makes team work much easy for software development since every developer know what others is doing and project leader can have better control of the progress. But there is also downsides for this type of setup, the most common and obvious one is the single point failure. When the centralized server is down, all member of the team cannot work anymore since no changes can be made to the files store in the server and it's also impossible to get code from the server.

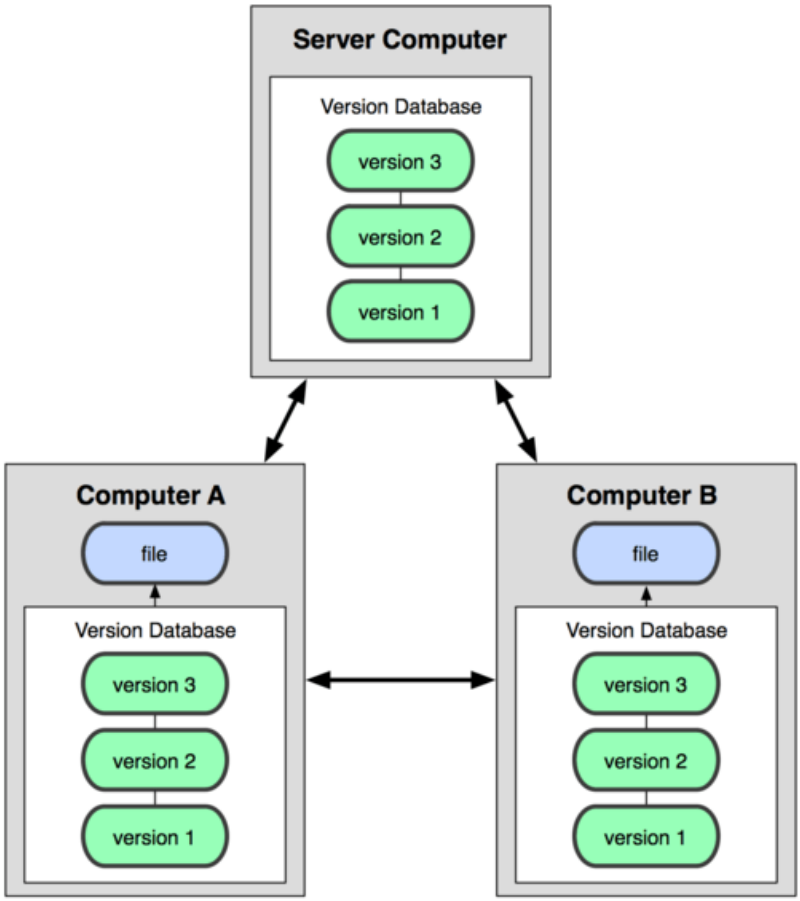


Figure 4-3: Distributed Version Control Diagram

This is why distributed version control repository is introduced in the last decade. Distributed version control repository takes a peer-to-peer approach as showed in Figure

4-3, which means when check out files, clients don't just check out the latest version of the files, but the whole repository. So that if any of the server downs, any of the client repository can be used as a server at once. This approach also causes some other important differences from a centralized system includes but not limit to:

There is not any reference copy any more, every copy in each repository is a working copy. Commits, history roll back and other common operations can be done locally without any communicate with other peers, which makes these operations much quicker than centralized setup.

There are a lot of version control repository available like CVS, PVCS, MKS, and Git etc. In this project, I decide to use Git [11]. Git is a free open source distributed version control repository initially designed for Linux kernel development in 2005. It is distributed under the term of GNU General Public License version 2. A survey of Eclipse IDE users shows that Git have 30% adoption as of 2013, which makes Git one of the most popular open source distributed version control repository today.

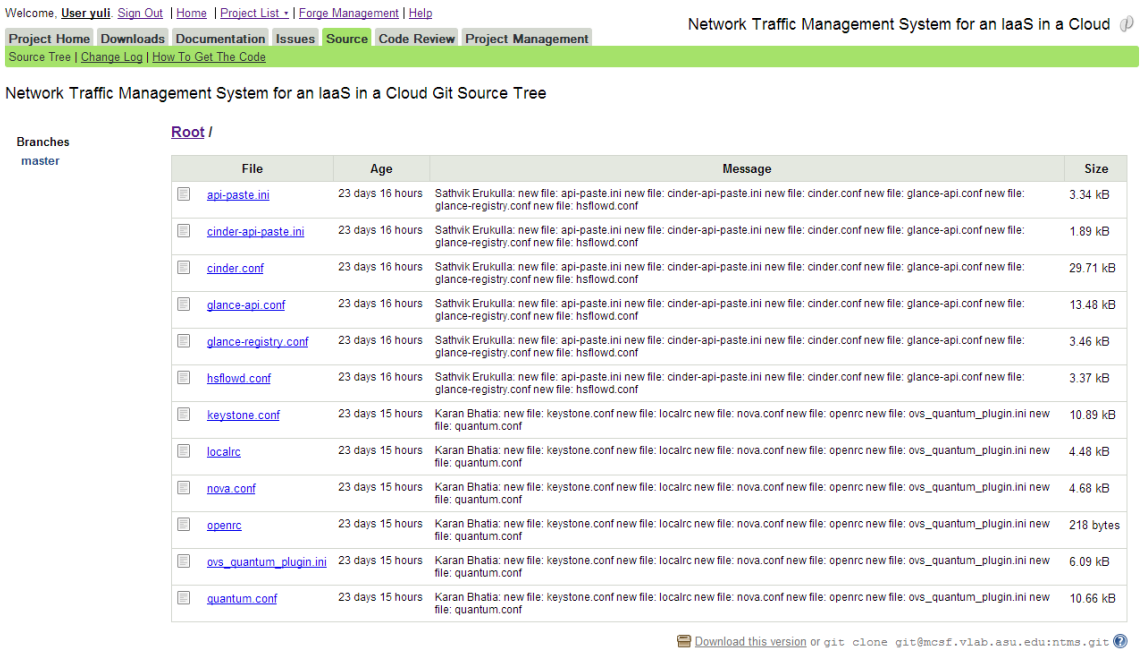


Figure 4-4 Web UI for Git Version Control Repository

A Web UI [12] for Git is also introduced into the system as shown in Figure 4-4.

This portal provides various options like Git account management and access control, project wiki site with tutorials, source code viewing and downloads, etc. Compare to traditional command line based Git interface, this UI not only provides software developer with a more user-friendly environment but also makes the management of source code much easier and clear.

4.1.2 Continuous Integration Server

Once we have our code stored in a repository, we need to use these codes to do integration. According to the Continuous Integration concept, this should be done by some automated build mechanism. Originally software developers intend to run script based integration tool on a build machine to do the continuous integration. Later, Continuous Integration Machine

was introduced. A CI machine is a standalone host which runs integration builds and unit tests periodically when a change of code is committed to the repository and report the results to the developers.

Since this project aims to provide the Continuous Delivery in a Cloud environment, instead of using a local based Continuous Integration Machine, a Continuous Integration Server is used. A Continuous Integration Server is a web server that running all the service provided by Continuous Integration Machine in Internet, and, instead of provide developer with command line on local machine, it provide developer a web interface to manage the whole continuous integration process.

In an integration, there are mainly two tasks, compile and test. Both of these two works was triggered by one Continuous Integration Server every time when there is a commit, and done by two different subcomponents in Continuous Integration Server introduced in the later section.

Other function provided by CI server are user account management, project configuration and management, etc.

4.1.3 Automated Build and Test System

As the main component of CI server and a Continuous Delivery system, automated build and test system basically handled two task for the CI server.

First is the compile and integration part, this part is handled by automated environments for builds. Automated environments for builds have been available for different types of systems for decades, like the make file for UNIX system, Ant developed for Java and

MSBuild for .NET platform. All these automated systems runs according to scripts written by developers and do integration step by step.

Then is the testing part, most of this part is handled by XUnit frameworks [13], including unit testing tools like JUnit, NUnit and so on. These tools allow developers to done automated testing by writing testing scripts and run a single command.

The command point of the compile and testing part are obvious. They're both totally automated and running on scripts preset by developers. This is because of that all the process include download dependency from Internet, compilation, loading database schemas, doing unit test and so on, through sometimes complicated, can all be automated. These processes also should be automated since doing this time by time manually is wasting time and it's also easy for people to make mistake in these processes.

Since this project aims to provide the Continuous Delivery in a Cloud environment, instead of using a local based Continuous Integration Machine, a Continuous Integration Server is used. A Continuous Integration Server is a web server that running all the service provided by Continuous Integration Machine in Internet, and, instead of provide developer with command line on local machine, it provide develop a web interface to manage the whole continuous integration process.

4.1.4 Build Script

As shown in section 4.1.3, build script is a very important part of Continuous Delivery, which make the whole system automated. The build script is a single script, or set of scripts, which developer uses to compile, test, inspect, and deploy software.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<project name="brewery" default="all" basedir=".">
  <target name="clean" />
  <target name="svn-update" />
  <target name="all" depends="clean,svn-update"/>
  <target name="compile-src" />
  <target name="compile-tests" />
  <target name="integrate-database" />
  <target name="run-tests" />
  <target name="run-inspections" />
  <target name="package" />
  <target name="deploy" />
</project>
```

Figure 4-5: Example Ant Script Shell

Figure 4-5 shows a simplified Ant script shell which will perform an integration. In the example script, it lists the steps that need to be done: Clean the build field; get the newest commit from SVN server; download resources and dependencies; compile the sources file and test file; integrates database needed; run tests and inspections; then last step is to package the software and deploy it.

More example of scripts will be provided in the usage scenarios section.

Build scripts are often particular to a platform, like most java projects use Ant and .NET project use MSBuild. But there are also some cross platform scripts. For example, Ruby and Maven will works for C, C++, Java and much so on. Since this project is aiming to provide instructors and students a tool with high usability, multiple scripts (with broad compatibility) support is installed in the CI server.

4.1.5 Feedback Mechanism

Feedback is also one of the key component of Continuous Delivery.

Since Continuous Delivery is a totally automated process, information about the result of every integration is very important to developers. Developer wants to know as soon as possible if there was a problem with the latest build. Widely used feedback mechanism include email, Short Message Service and Really Simply Syndication.

In this project, email mechanism provide by the CI server is used.

4.2 Usage Scenarios

This section presents usage scenario reproduces real life software development scenes. These scenarios will show us how the components described in previous sections working together in education field to achieve the proposed goal of this research.

4.2.1 A Research Project Scenario

In this scenario, a team of graduate students trying to use the proposed system to develop a Java program for research purpose. This is going to be a yearlong project which will involve multiple students as developers. These students are going to work separately at most of the time. During this scenario, there will be two main stages: Project setting up stage and project developing stage.

During the project setting up stage, most of the work needs to be done by a project leader. Project leader need to access to CI server to setup a new project for the team. Figure 4-6 and 4-7 shows screenshots of the project setting up pages of the CI server.

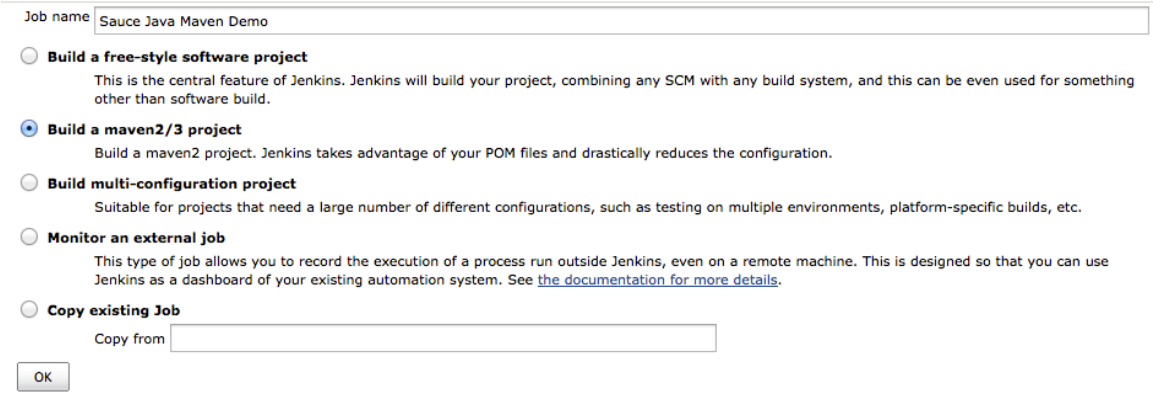


Figure 4-6: New Project Page of CI server

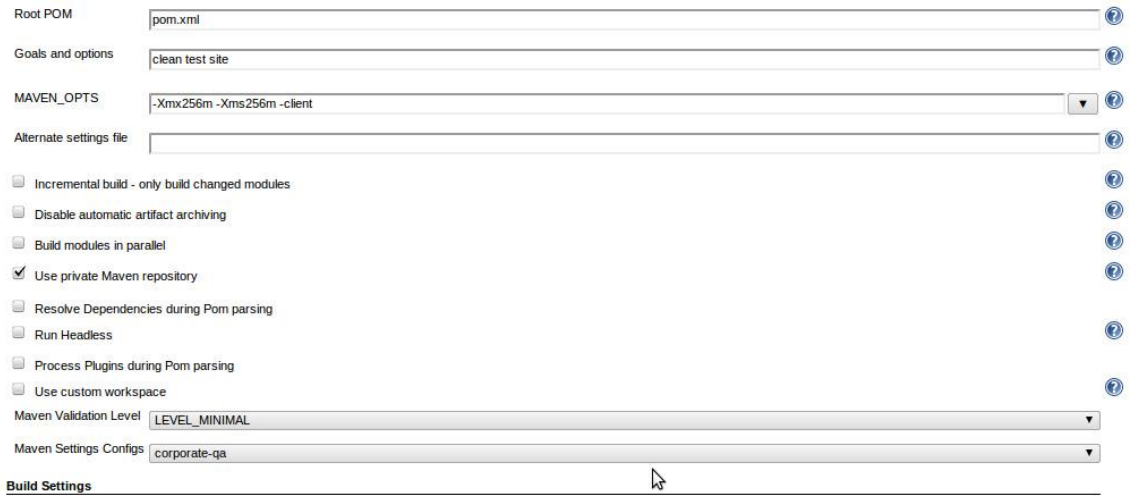


Figure 4-7: Maven Project Setting Page of CI server

Since this is a Java project, this team decides to use Maven [14] as the auto automated build and test system, which is already support by the CI server. Other available choice support by CI server for java project includes Ant. To use Maven as a build system for this project, an xml script called Project Object Model (POM) must be provided to the CI server. Figure 4-8 show a minimal example of a POM file.

```
<project>
  <!-- model version is always 4.0.0 for Maven 2.x POMs -->
  <modelVersion>4.0.0</modelVersion>

  <!-- project coordinates, i.e. a group of values which
       uniquely identify this project -->

  <groupId>com.mycompany.app</groupId>
  <artifactId>my-app</artifactId>
  <version>1.0</version>

  <!-- library dependencies -->

  <dependencies>
    <dependency>

      <!-- coordinates of the required library -->

      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>

      <!-- this dependency is only used for running and compiling tests -->

      <scope>test</scope>

    </dependency>
  </dependencies>
</project>
```

Figure 4-8: POM Example for Maven Project.

Once the POM file is provided to CI server, the project setting up stage is done. One thing needs to be noticed is that for a complicated Java project with lots of different dependencies, POM file will be very complicated. It's common to see a POM file contains more than 1000 lines of code. Through there are generate tools can automatically generate some part of the POM file, manual coding is still essential.

Then it go through into project developing stage, this is where the Continuous Delivery actually happened. Figure 4-9 shows the sequence diagram of the whole process.

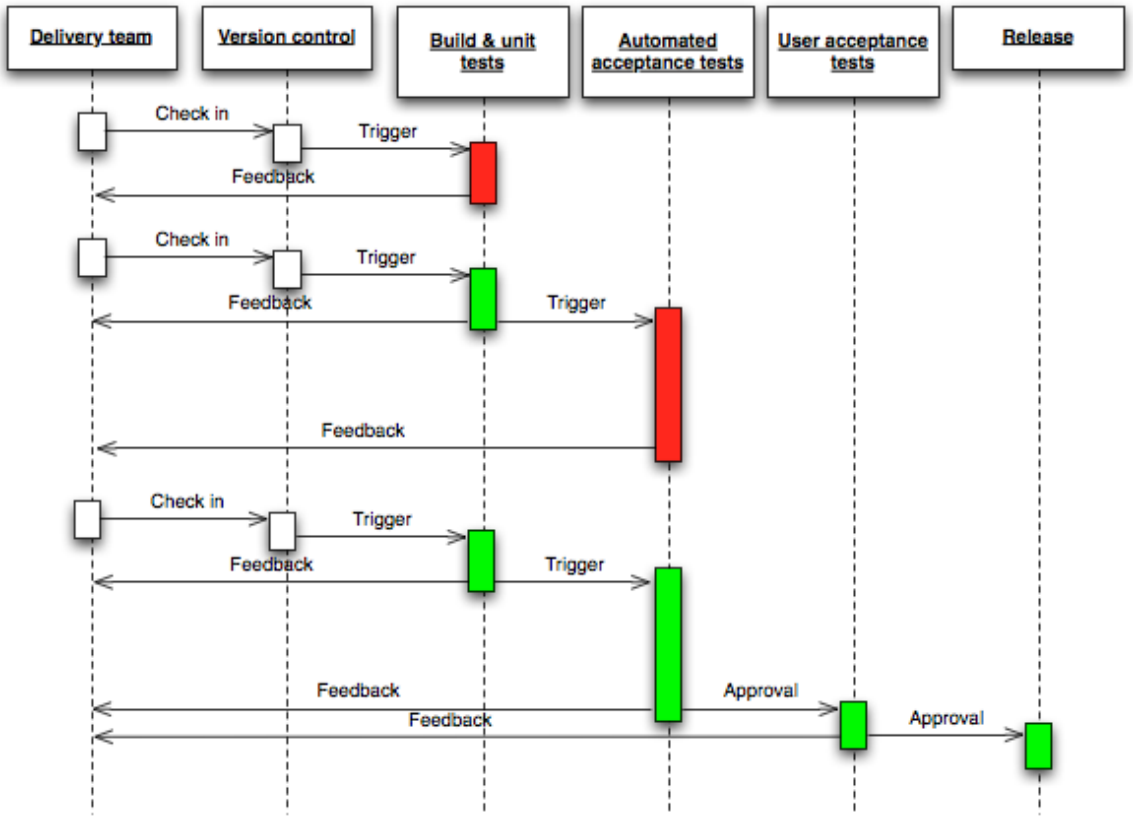


Figure 4-9: Sequence Diagram of Software Developing Stage

First, a student checks in his new source code into the version control system, which triggers the Maven system in the CI server to run build and unit tests. But, during the build and unit test procedure, some error occurred. Build error message and unit tests report are generated by the Maven system and sent to the student who commit the code and project leader immediately by CI server.

After some debugging, the student fixed the low level bug in his code and checks in source code into the version control system again. This time, the code goes through the build and unit test procedure. Notified by Maven, CI server then triggers the automated acceptance test mechanism.

In automated acceptance test mechanism, there are two procedures. First is the provisioning of test and staging environment. To construct a virtual machine and provision it for test and staging, CI server first contacts with Vlab platform to load an image and start an instance with that image, after that CI load a system configuration script to provision the instance provided by Vlab platform. Below is an example of a system configuration script which installs MySQL database into the test instance:

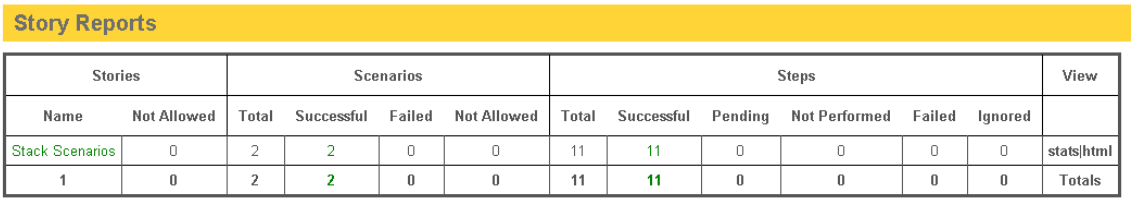
```
class mysql {
 $mysql_password = "p4ssw0rd"

 package { "mysql-server":
   ensure => installed
  }

 package  {"mysql":
   ensure => installed
  }

 service { "mysqld":
   enable => true,
   ensure => running,
   require => Package["mysql-server"],
  }
```

```
  exec { "set-mysql-password":
    unless => "mysqladmin -uroot -p$mysql_password status",
    path => ["/bin", "/usr/bin"],
    command => "mysqladmin -uroot p4ssw0rd $mysql_password",
    require => Service["mysqld"],
  }


define  mysql::db() {
    include mysql
        exec { "create-${name}-db":
        unless => "/usr/bin/mysql -uroot -pp4ssw0rd ${name}",
        command => "/usr/bin/mysql -uroot -pp4ssw0rd -e \"create database ${name};\"",
        require => [Service["mysqld"],Exec[set-mysql-password]]
       }
}


define mysql::createuser($user, $password, $db, $host) {
      include mysql
      exec { "grant-${name}-db":
        unless => "/usr/bin/mysql -u${user} -p${password} ${name}",
        command => "/usr/bin/mysql -uroot -pp4ssw0rd -e \"grant all on *.* to
${user}@'${host}' identified by '$password' WITH GRANT OPTION; flush privileges;\"",
```

```
      require =>  Mysql::Mysql::Db[worblehat]
    }
  }
}
```

The system configuration script is written in very simple languages and is easy to understand. We can see what this script does. First, it ensures MySQL and MySQL server is installed on the Vlab instance and is running. Then it sets the root user password for the MySQL server, creates a new database in MySQL and finally creates a new user.

Once the provisioning of test and staging environment is done, the automated acceptance test mechanism continuous into next procedure: deploy the software integration result into the provisioned testing instance and do the automated acceptance tests. The result of the automated acceptance tests will be generated by the CI server and report to students.



Story Reports

Stories		Scenarios				Steps						View
Name	Not Allowed	Total	Successful	Failed	Not Allowed	Total	Successful	Pending	Not Performed	Failed	Ignored	
Stack Scenarios	0	2	2	0	0	11	11	0	0	0	0	stats|html
1	0	2	2	0	0	11	11	0	0	0	0	Totals

Figure 4-10: An Example of an Automated Acceptance Tests Report

Figure 4-10 show a simple example of an automated acceptance tests report generated by CI Server automatically which shows the software passed all the tests. The automated acceptance tests is also done by CI server according to some self-testing script.

Below is a simple example of a self-testing script for testing java program:

```
@Given(“an empty stack”)
   public void anEmptyStack() {
```

```
    testStack = new Stack<String>();
}

@When("the string $element is added")
public void anElementIsAdded(String element) {
    testStack.push(element);
}

@When("the last element is removed again")
public void removeLastElement() {
    testStack.pop();
}

@When("the element $element is searched for")
public void searchForElement(String element) {
    searchElement = element;
}

@Then("the resulting element should be $result")
public void theResultingElementShouldBe(String result) {
        Assert.assertEquals(testStack.pop(), result);
}
```

```
@Then(“the position returned should be $pos”)
public void thePositionReturnedShouldBe(int pos) {
    Assert.assertEquals(testStack.search(searchElement), pos);
}
```

It's easy to tell that this simple script does some basic behavior test with a stack data structure implemented in the software.

Go back to the sequence diagram. For this time, the code submitted by the student didn’t pass the automated acceptance test. Feedback as well as the test report are sent back to the students immediately.

Once again, the students do some debugging according to the report he got and do some more changes to the code before recommit it.

After the last commit done by the student, the project runs smoothly from unit test to automated acceptance test. Sometime later that day, it also passes user acceptance test done manually by other team members and is ready to be released.

One thing needs to be noticed is that the thing happens in that sequence diagram will be done by all the team member of the research group every day. This keeps evolving the software and also makes sure the software is ready to be released all the time.

CHAPTER 5

SYSTEM EVALUATION

To implement the Continuous Delivery software developing system, we established the system on Vlab platform.

5.1 Virtualized Hardware

As Figure 4-1 shows, the architecture of the system is consisted by three parts: Version control repository, CI server and Production Environment. All these three part are deployed in Vlab platform in three separate virtual machines.

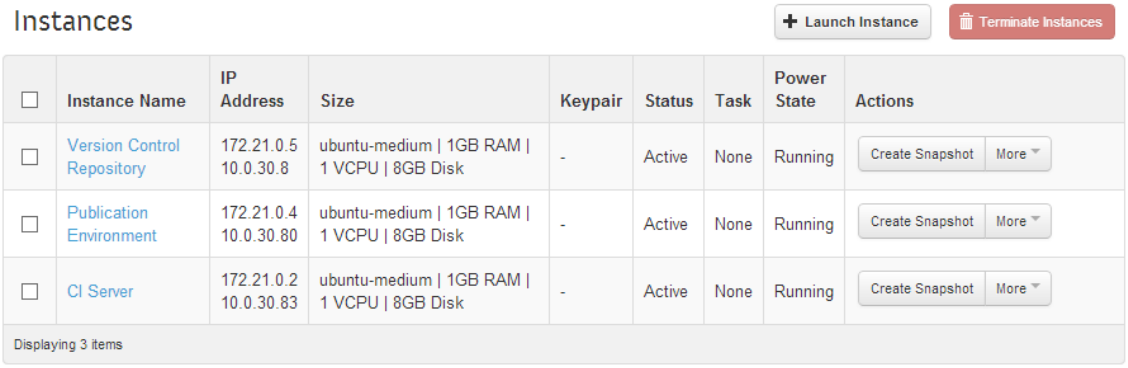


Figure 5-1: Instances List in Vlab Dashboard

Figure 5-1 shows a list of running instances in Vlab platform in my project, we can see each instance have the same virtualized setup: 1 virtual CPU, 1GB of virtual memory and 8GB of virtual disk storage and running Ubuntu 12.04 OS. All these three instance are running on the same Dell PowerEdge R620 rack-server, which has 24 cores 2.40 GHz processors, 64GB of memory and 2TB of storage space. Another iSCSI SAN storage array is attached to the rack-server to provide extra storage space. By virtualization, the rack-server can provide 80 virtual CPU and 48GB of virtual memory to all the instance running

on it. Currently the CPU load of the rack-server is low than 40% all the time with a total of 42 instances running on it.

5.2 Network Configuration

In my implementation, all three instances running system components are connect to the same 172.21.0.X network within the virtual network environment in the rack-server. Using this network, three instances can communicate with each other directly. Two make it accessible from outside of the virtual network environment, a floating IP in 10.0.30.X network is assigned to each of the instance. This network is a physical network communicating across the Vlab platform. The web gateway of the Vlab platform is configured to forward traffic between the Vlab platform internal network and Internet.

Two virtual host proxy has been setup for CI server instance and version control repository instance in the gateway. Http://mcsf.vlab.asu.edu has been assign to version control repository, and http://mcad.vlab.asu.edu has been assign to CI server.

Both of these two instance are available for access through Internet.

5.3 User Access Control

Identity management of Vlab provides a single point user account management and authentication service for the proposed system. The current Vlab system combined LDAP and Identity to provide identity service to other components of cloud.

Students are required to sign up for a Vlab user account to access the proposed system. Then each component handles role-based access control by themselves. As a result, students can have different roles in different components. One may be a project leader of

one project in version control repository and be normal developer for the same project in the CI server. Furthermore, to provide more secure access control for the version control repository, only SSH based access is allowed when communicate with version control server through Git protocol. Students are required to generate public/private key pairs and upload their public keys into version control system through web portal before they can commit source code to version control system or pull codes from the system.

5.4 Performance Evaluation

In this section, we compared the proposed system with a PC.

PC test bed is a ThinkPad T430s with Intel Core I5 CPU and 4GB of Memory running Ubuntu 12.04 and Eclipse IDE.

The PC and the system were set to compile the same software using the same build tools and the time costs were recorded for evaluation.

There are two sets of java source code used in this evaluation. One is a basic ‘hello world’ program and another one is a calculator with basic GUI which has about 600 lines of codes and some dependence. Both of the project are configured to be run by Maven tool. In PC test bed, Eclipse IDE with Maven plugin was used to compile the program.

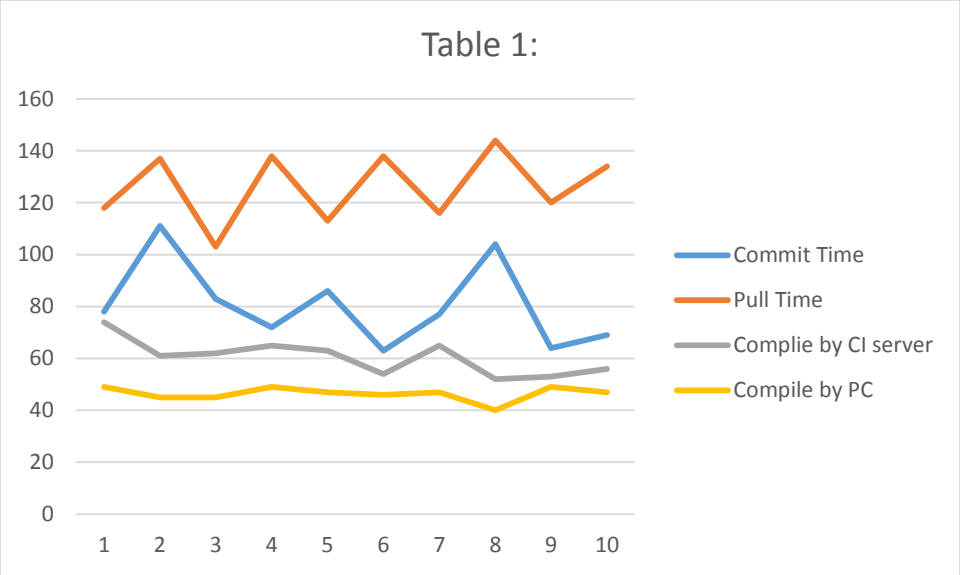


Figure 5-2: 'Hello World' Program Operation Time

Operation	**Average Time /ms**
Commit Code to Version Control System	133.8
Pull code from Version Control System	92.1
Compile using CI server	64.7
Compile using PC	42.8

Table 5-1: 'Hello World' Program Operation Average Time

From Figure 5-2 and Table 5-1 we can see that for this kind of small program, time cost of commit and pull operation change significantly according to the network statues but the cost is relatively small. And time cost for compile is comparable between CI server and PC.

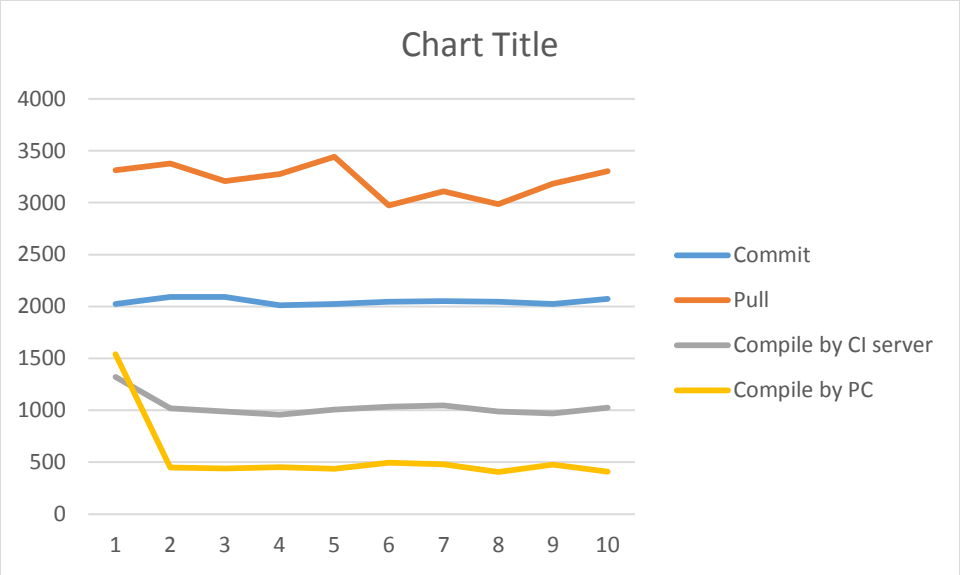


Figure 5-3: Calculator Program Operation Time

Operation	**Average Time /ms**
Commit Code to Version Control System	2048.4
Pull code from Version Control System	3217.3
Compile using CI server	1035.7
Compile using PC	558.3

Table 5-2: Calculator Program Operation Average Time

In Figure 5-3, we notice that it took longer time for both PC and CI server to compile for the first time. This is because that the Maven tool needs to download some dependence online before compiling. And as a result of a better Internet connection, CI server finished the first compiling even quicker than PC, which is very impressive.

From Figure 5-3 and Table 5-2, we can see that pull operation requests more time and more relies on Internet connection statues when there have been a lot of change made to the

version control system. CI server do takes more time to do compiling than PC as a result of the computing power difference, but the gap is not very big.

CHAPTER 6

CONCLUSION

In this chapter, the summary of current work and the plan of future work will be discussed.

6.1 Conclusion of Current Work

Research presented in this thesis attempts to adopt the Continuous Delivery concepts on to Vlab platform to provide instructors and students in Arizona State University with an automated software development infrastructure with low learning curve and high availability based on Cloud environment.

To achieve this result, this research proposed Continuous Delivery system architecture and design base on Vlab platform to overcome barriers preventing Continuous Delivery to be introduced into education field.

To overcome the problem of long initial setup, we implement a well-designed and dynamic changing Continuous Delivery software development system to students directly from Internet. And, to allow students to take advantages of the automated testing mechanism of Continuous Delivery, we proposed a project template-based solution which highly reduces the learning curve of the automated testing mechanism and also provides instructors a useful teaching tool. Last but not least, by implementing the system in an existing Cloud environment, the initial setup hardware cost is reduced to zero, which makes it possible for us to provide this system as a service to Arizona State University instructors and students totally free.

6.2 Future Work

In further research, a centralized role-based authentication and authorization system needs to be developed and introduced to current system. This system can provide single sign-on, cross-components role control and a lot of other benefits currently unavailable.

Also, a software publication environment with web UI and mobile phone application is in our development plan. This will provide students with a place to publish their software development achievement to public.

A service oriented system modification is also under design. Our next goal is to implement a Cloud based web service development, publish, discovery and integration system to make collaboration between research groups much easier.

REFERENCES

[1] Agile software development page on Wikipedia, http://en.wikipedia.org/wiki/Agile_software_development

[2] Continuous Delivery page on Wikipedia, http://en.wikipedia.org/wiki/Continuous_delivery

[3] A Kadne, *Vlab: A Cloud based Resource and Service Sharing Platform for Computer and Network Security Education*, Master Thesis, 2010.

[4] The ThoTh, https://vlab.asu.edu/

[5] Degree programs in Software Engineering, http://www.gradschools.com/search-programs/software-engineering

[6] *OpenStack Cloud Administrator Guide.* OpenStack Foundation. http://docs.openstack.org/admin-guide-cloud/content/ch_getting-started-with-openstack.html

[7] C Caum, *Continuous Delivery vs. Continuous Deployment*, Puppet Labs Blog, 2013. http://puppetlabs.com/blog/continuous-delivery-vs-continuous-deployment

[8] M Flowler, *Continuous Integration*, 2006. http://martinfowler.com/articles/continuousIntegration.html

[9] Concurrent Versions System page on Wikipedia, http://en.wikipedia.org/wiki/Concurrent_Versions_System

[10] Aljord, P., & Danic, M. *Pro Git.* Apress Media LLC. 2007. ISBN 978-1430218333

[11]Git website, http://git-scm.com/

[12] Indefero project by C éonde Ltd website, http://projects.ceondo.com/p/indefero/

[13]XUnit page on Wikipedia, http://en.wikipedia.org/wiki/XUnit

[14] Apache Maven website, http://maven.apache.org/

[15] Apache JMeter website, http://jmeter.apache.org/

[16] Duvall, P. *Continuous Integration: Improving Software Quality and Reducing Risk.* Addison-Wesley, Upper Saddle River 2007. ISBN 978-0321336385

[17] Farley, D. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Addison-Wesley, Upper Saddle River 2011. ISBN 978-0321601919.

APPENDIX A

VLAB PLATFORM

This appendix briefly describes each components of current Vlab platform as an additional material of Chapter 2 of this thesis. We refer to [6] hugely to summarize the features implemented in Vlab platform. If only interested in understanding the design and workflow of the proposed system, readers can simply skip this appendix.

A.1 Dashboard of Vlab

This Dashboard provides end users and cloud administrators with interface to all other OpenStack services.

This provide end users with interface of services like usage information, cloud virtual machine operation, management of block storage space, image and snapshot management to upload, delete and control virtual images, and access control to manage key pairs and firewall rules of network.

Other than these, Dashboard also provides extra interface for cloud administrators which allow administrator to define service catalog offerings of resources, manage project setting, administer user accounts and view services running in the cloud.

A.2 Compute of Vlab

Nova is the most important component of the whole cloud. It’s the place where all API requests turned into running virtual machines. It’s also the most complicated part of the system with a lot features. Among all the provided features, the most important features are starting, stopping and querying virtual machines, assigning and removing public IP addresses, attaching and detaching block storage, adding, modifying and deleting security groups, show instance consoles and snapshot running instances.

A.3 Image Store of Vlab

Image Store stores virtual images for users and other cloud services.

It has the following functionalities: Stores public and private images that users can utilize to start instances, query and list available images for use, delivers images to Compute to start instances and snapshots from running instances so that virtual machines can be backed up.

There are also a number of periodic process which run on Image Store to support caching. The most important of these is the replication services, which ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters and reapers.

A.4 Identity of Vlab

Identity of Vlab provides a single point for the whole cloud system which stores and manages policy, catalog, token and authentication.

It can authenticate users and issue tokens for access to services, store user and tenants for a role-based access control, provides a catalog of the services in the cloud and manage policies across users and services.

The current Vlab system combined LDAP and Identity to provide identity service to other components of cloud.

A.5 Network of Vlab

Last but not least, Network plays a very important roles in current Vlab platform. It provides network as a service (NaaS) between virtual machines running in the cloud

platform. It allow users to create their own networks and then attach virtual machine interfaces to them. It also has a lot of extensions that enable additional network services like load balancing. Network will interact mainly with Compute, where it will provide networks and connectivity for its instances.

Like other Vlab services, Network is highly configurable due to its plug-in architecture. These plug-ins accommodate different networking equipment and software. As such, the architecture and deployment can vary dramatically.